Artificial Intelligence

Contents

1 Definition

Artificial Intelligence studies how we can make the computer do things that humans can still do better at the moment

→ AI is decreasing

Aspects:

- Ability to learn
- Inference (drawing conclusions)
- Perception (Wahrnehmung der Umwelt)
- Language understanding (as communication in general)
- Emotion

Analysis			
Deep	Knowledge-based	Not there yet	
Shallow	No-one wants this	Statistical Methods	
	Narrow	Wide	**Coverage**

Precision			
100%	Producer Task		
50%		Consumer Tasks	
	$10^{3\pm1}$	$10^{6\pm1}$	**Coverage**

Narrow / weak AI = study or accomplish *specific* problem solving or reasoning tasks
Strong AI / AGI (Artificial general intelligence) = software performing at the full range of human cognitive abilities

Problems requiring strong AI to solve are called *AI complete.*

2 ProLog

Constants → lower-case
Variables → upper-case or underscore (never-used variable)
Functions and *predicates*
→ lower-case, applied to terms
→ Parameters have no unique direction "in" or "out"

Program:

- *Facts* → $t.$
- *Rules*
  → $h :- b_1, ..., b_n.$ with h the head literal and b_i the body literals
  ⇒ h if $b_1, ..., b_n$.

Knowledge base → set of facts that can be derived of the program
Query → $? - A_1, ..., A_n.$ with A_i terms called goals
Backchaining → Testing whether query is true or false

Search procedure:

- Top-down
- Left-right
- Depth-first: *Backtracking*

Internal arithmetic: ?- D is e.
→ e is a ground arithmetic expression that binds D to the result of

evaluating e

Lists

$\rightarrow$ [a, b, c, ...]

$\rightarrow$ [F|R] with F the first element and R the rest list

2.1 Functions

```
% remove duplicates function
delete(_,[],[]).
delete(X,[X|T],R) :- delete(X,T,R).
delete(X,[H|T],[H|R]) :- not(X=H), delete(X,T,R).
removeDuplicates([],[]).
removeDuplicates([H|T],[H|R]) :- delete(H,T,S), removeDuplicates(S,R).

% reverse function
preReverse([],X,X).
preReverse([X|Y],Z,W) :- preReverse(Y,[X|Z],W).
myReverse(A,R) :- preReverse(A,[],R).

% permute function
takeout(X,[X|T],T).
takeout(X,[H|T1],[H|T2]) :- takeout(X,T1,T2).
myPermutations([],[]).
myPermutations([X|Y],Z) :- myPermutations(Y,W), takeout(X,Z,W).

%zip
zip([L],[],[[L]]).
zip([],[L],[[L]]).
zip([A],[B],[[A,B]]).
zip([H1|T1],[H2|T2],L) :- zip(T1,T2,T), append([[H1,H2]],T,L).
```

```
add(X,nil,tree(X,nil,nil)).
add(X,tree(Root,L,R),tree(Root,L1,R)) :- X @< Root, add(X,L,L1).
add(X,tree(Root,L,R),tree(Root,L,R1)) :- X @> Root, add(X,R,R1).

construct(L,T) :- construct(L,T,nil).

construct([],T,T).
construct([N|Ns],T,T0) :- add(N,T0,T1), construct(Ns,T,T1).

count_leaves(nil,0).
count_leaves(tree(_,nil,nil),1) :- !.
count_leaves(tree(_,L,R),N) :- count_leaves(L,NL), count_leaves(R,NR), I

symmetric(nil).
symmetric(t(_,L,R)) :- mirror(L,R).

mirror(nil,nil).
mirror(t(_,L1,R1),t(_,L2,R2)) :- mirror(L1,R2), mirror(R1,L2).
```

DFS:

```
% dfs(SearchedValue, Tree, Path, Cost)
dfs(GoalValue,tree(GoalValue,_),GoalValue,0).

dfs(GoalValue,tree(Value,[(Cost,T)|Rest]),Path,FinalCost):-
        T = tree(IV,_), write(IV ),
        dfs(GoalValue,T,P,C),
        string_concat(Value,P,Path),
        FinalCost is C+Cost; % go down one depth level
        dfs(GoalValue,tree(Value,Rest),Path,FinalCost).
        % next child
```

BFS:

```
% helper method to build up a fringe - it takes the Children
%of a tree, inserts the path to its parent into the touple
%and sums up the cost
insert(_,_,[],[]).

insert(Path,Cost,[(C,T)|Rest],[(T,Path,NC)|NewRest]) :-
        NC is Cost+C,
        insert(Path,Cost,Rest,NewRest).

% bfs(SearchedValue, Fringe, Path, Cost),
% where Fringe is a list of touples (Node,PathToNodesParent,TotalCostToNode)
bfs(GoalValue,[(tree(GoalValue,_),OldPath,FinalCost)|_],FinalPath,FinalCost):-
        string_concat(OldPath,GoalValue,FinalPath),
        write(GoalValue).
```

```
bfs(GoalValue,[(tree(Value,Children),Path,Cost)|Fringe],FinalPath,FinalCost):-
        write(Value),
        string_concat(Path,Value,NewPath),
        insert(NewPath,Cost,Children,NewChildren),
        append(Fringe,NewChildren,NewFringe),
        %btw. changing Fringe and NewChildren here would make this into dfs
        bfs(GoalValue,NewFringe,FinalPath,FinalCost).

bfs(GoalValue,Tree,Path,Cost) :-
        istree(Tree),
        bfs(GoalValue,[(Tree,"",0)],Path,Cost).
```

3 Complexity

→ Worst-case time/space complexity

- Constant: $\mathcal{O}(1)$
- Logarithmic: $\mathcal{O}(ln(n))$
- Linear: $\mathcal{O}(n)$
- Quadratic: $\mathcal{O}(n^2)$
- Polynomial: $\mathcal{O}(n^k)$
- Exponential: $\mathcal{O}(k^n)$

P = alle Probleme, die deterministisch in Polynomialzeit lösbar sind

NP = alle Probleme, die von nicht-deterministischen Turingmaschinen in Polynomialzeit lösbar sind

$P \subset NP$
$P = NP$???

4 Intelligent Agents

AI definitions:

1. Acting humanly
   $\rightarrow$ Turing test
2. Thinking humanly
   $\rightarrow$ Cognitive Science and Cognitive Neuroscience
3. Thinking rationally
   $\rightarrow$ Aristotle
4. Acting rationally
   $\rightarrow$ Acting so that you would expect to maximize your goal achievement + thinking involved

An agent a ($f_a : \mathcal{P}^* \rightarrow \mathcal{A}$) perceives his environment via sensors ($\mathcal{P}$) and acts on it ($\mathcal{A}$) with actuators

An agent function

- specifies the input-output relation (outside view)
- takes the full sequence of percepts as arguments

An agent program

- implements the function (inside view)
- uses the internal state to avoid the full sequence of percepts
- there are either none or infinitely many programs for a function

A performance measure is a function that evaluates a sequence of environments

An agent is called rational if it chooses whichever action maximizes the *expected value* of the performance measure given the percept sequence to date

- Rational $\neq$ hellsichtig $\rightarrow$ only maximize expected value
- Rational $\neq$ allwissend $\rightarrow$ percepts may not supply all relevant data

  $\rightarrow$ but try to explore best
- Rational $\neq$ successful

  $\rightarrow$ but try to learn best

Rational = exploration, learning, autonomy

An agent is called autonomous if it does not rely on the prior knowledge of the designer

Describing the tasks environment:

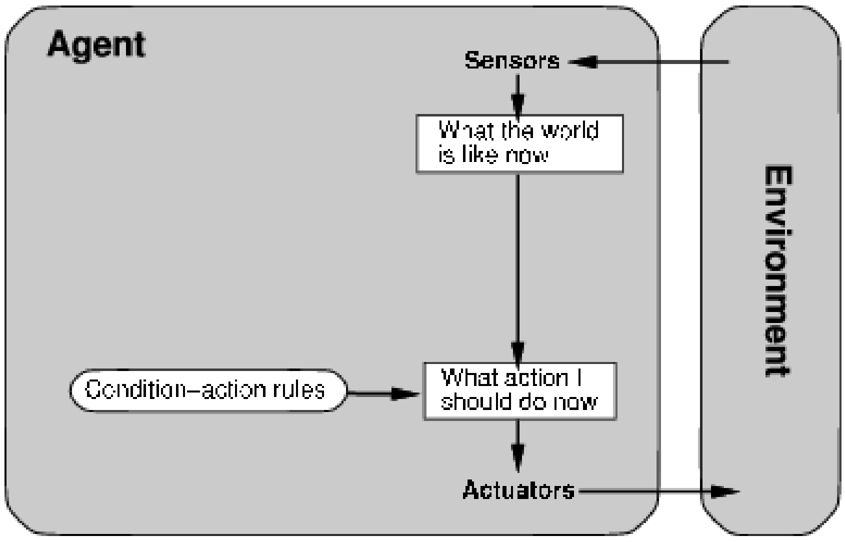
- Performance measure
- Environment
- Actuators
- Sensors

Environment types

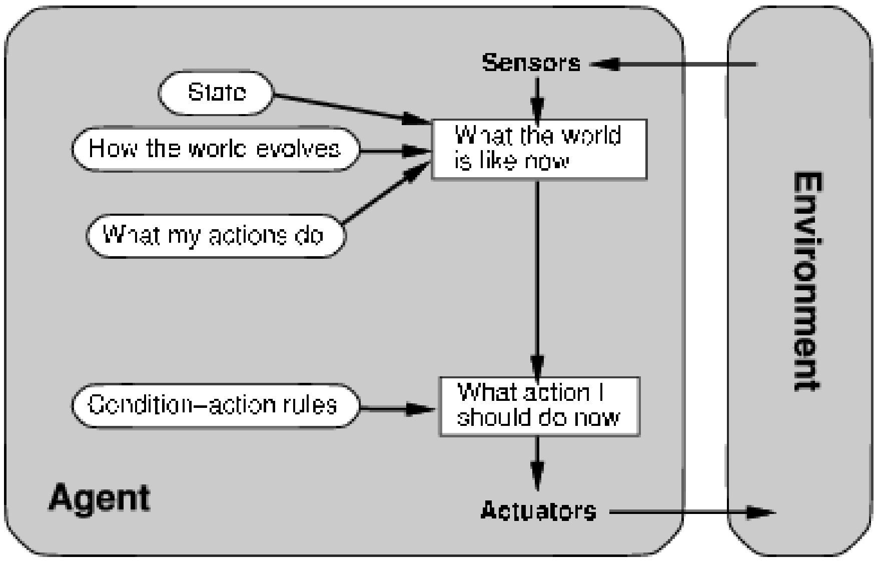
- Fully observable $\leftrightarrow$ partially observable
- Deterministic $\leftrightarrow$ stochastic
- Episodic $\leftrightarrow$ sequential (state depends on previous state)
- Dynamic $\leftrightarrow$ semidynamic (only performance measure changes) $\leftrightarrow$ static (nothing changes without the agent doing something)
- Discrete (states, actions are countable) $\leftrightarrow$ continuous
- Single-agent $\leftrightarrow$ multi-agent

Agent types

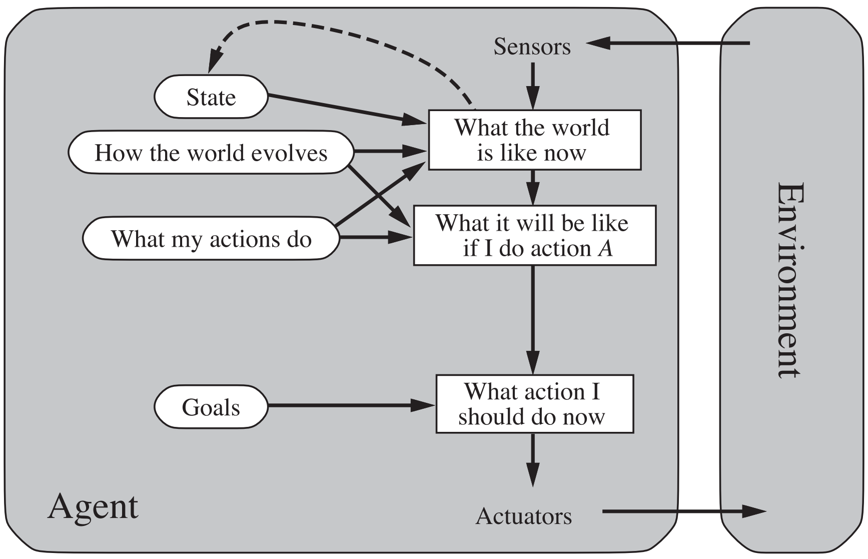
- Simple reflex agent
  → actions only base on the last percept



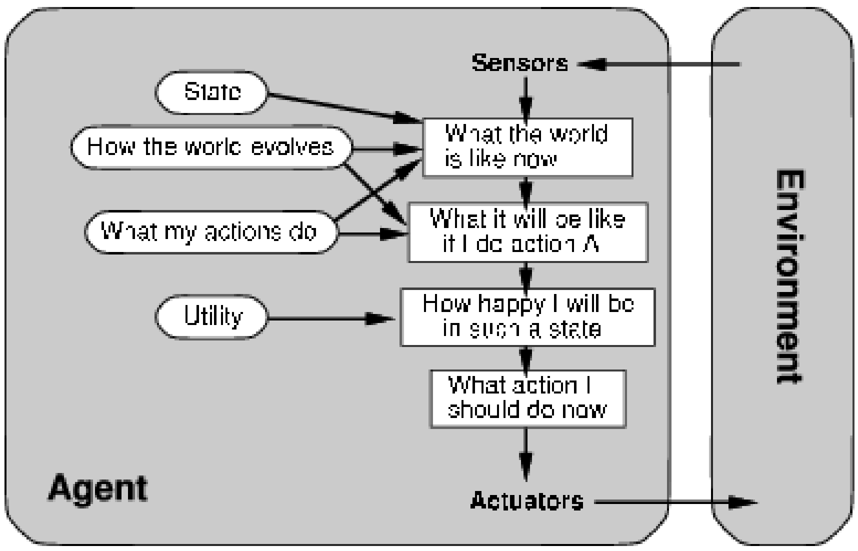
- Reflex agent with states



- Goal-based agent
  → is a stateful reflex agent with a goal

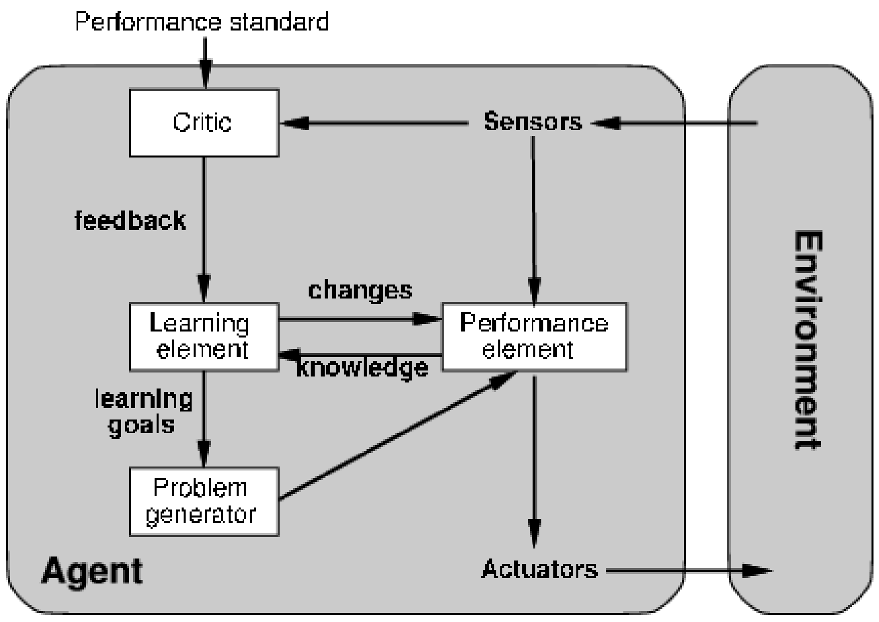

Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

- Utility-based agent
  → A utility-based agent uses a worldmodel along with a utility function that influences its preferences among the states of that world. It chooses the action that leads to the best expected utility, which is computed by averaging over all possible outcome states, weighted by the probability of the outcome


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Utility
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

- Learning agent (All the above + learning)
  → ameliorates the performance measure
  - Learning element → improving the agent's knowledge
  - Critic → gives feedback on learning element based on external performance standard

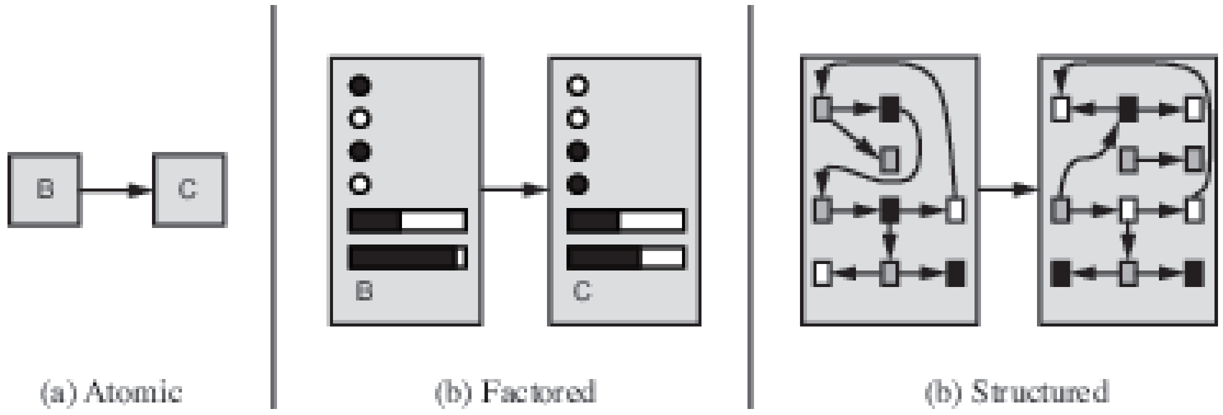
  – Problem generator → suggests action leading to new, informative experiences
  – Performance element = Agents without learning



Domain-Specific Agent ↔ General Agent

State Representation:

- Atomic → No internal structure
- Factored → Each state is characterized by attributes and their values
- Structured → State includes objects and their relations



5 Problem Solving and Search

Problem = States + Actions

Offline problem solving $\leftrightarrow$ Online problem solving

Problem formulation:

- Search problem: $\mathcal{P} := \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$
- States: $\mathcal{S}$
- Operators: $\mathcal{O} \subseteq \mathcal{S} \times \mathcal{S}$
- Goal States: $\mathcal{G} \subseteq \mathcal{S}$
- Initial State: $\mathcal{I}$
- Cost function: $c : \mathcal{O} \to \mathbb{R}_0^+$
- Step cost: $c(o) \quad o \in \mathcal{O}$
- Actions (Operator application): $s \to^o s'$, if $o = (s, s') \in \mathcal{O}$ with s' the successor of s
- Goal test
- Solution: Sequence of operators that bring us from $\mathcal{I}$ to $\mathcal{G}$

Problem description:

- Blackbox description $\to$ Algorithm has no information about the problem
- Declarative description $\to$ describes the problem itself (problem description language)

Problem types:

- Single-state problems
  - $\rightarrow$ Observable, deterministic, static, discrete
- Multiple-state problems
  - $\rightarrow$ Initial state not observable or partially observable, deterministic, static, discrete
- Contingency problems
  - $\rightarrow$ non-deterministic, unknown state space

Tree Search Algorithms

$\rightarrow$ Make a tree out of the graph

$\rightarrow$ Offline algorithm

$\rightarrow$ Search strategy = function that picks a node from the fringe of a search tree

Properties of Strategies:

$\rightarrow$ Completeness = does it always find a solution?

$\rightarrow$ Time complexity = number of nodes expanded

$\rightarrow$ Space complexity = number of nodes in memory

$\rightarrow$ Optimality = does it always find a least-cost solution?

b = maximum branching factor
d = minimal depth of a solution
m = maximum depth of the search tree

5.1 Uninformed Search Strategies

5.1.1 Breadth-first search

$\rightarrow$ Fringe is a FIFO queue

- Complete (if b is finite)
- Time $\mathcal{O}(b^{d+1})$
- Space $\mathcal{O}(b^{d+1})$ (keeps all nodes in memory)
- Optimal if cost = 1 per step

5.1.2 Uniform-cost search

$\rightarrow$ Fringe is queue ordered by increasing path cost (if equal cost FIFO)
$\rightarrow$ Add paths costs from the precessor node to the path cost of the current node

- Complete (if step costs > 0)
- Time: # nodes with path-cost less than that of optimal solution
- Space: # nodes with path-cost less than that of optimal solution
- Optimal

5.1.3 Depth-first search

$\rightarrow$ Fringe is a LIFO queue

- Complete if state space is finite (no loops or infinite paths)
- Time $\mathcal{O}(b^m)$
- Space $\mathcal{O}(b \cdot m)$ (keeps all nodes in memory)
- Not optimal

5.1.4 Iterative deepening search

```
$for(depthLimit = 0;\ depthLimit < TreeHeight;$
$depthLimit++)\{$
$depthFirstSearch(TreeCutByDepthLimit)$
$\}$
```

$\rightarrow$ Always starts again from the root

- Complete
- Time $\mathcal{O}(b^{d+1})$
- Space $\mathcal{O}(b \cdot d)$ (keeps all nodes in memory)
- Optimal if step cost $= 1$

5.2 Informed Search Strategies

$\rightarrow$ introduce information from outside the problem

5.2.1 Best-first search

Sort the fringe by an evaluation function
$\rightarrow$ expanding the most desirable node first
$\rightarrow$ Examples: Greedy Search, A^* Search

5.2.2 Heuristics

$\rightarrow$ Function that estimates the cost from the current node to the nearest goal state

- $h : S \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$ so that $h(s) = 0$ with s a goal state
- Goal distance function: $h^* : S \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$ where $h^*(s)$ is the cost of a cheapest path from s to a goal state or ∞ if no such path exists

Properties:

- **Admissible**: $h(s) \leq h^*(s)$ for all $s \in S$
- **Consistent**: $h(s) - h(s') \leq c(o_{s,s'})$ for all $s \in S$ and $o \in O$
- **Consistency implies Admissibility**

h_2 *dominates* h_1 if $h_2(n) \geq h_1(n)$ for all n
$\rightarrow$ If h_2 dominates h_1, then h_2 is better for search than h_1

$\Rightarrow$ Find a heuristic for a relaxed problem ($\mathcal{P}^r := \langle \mathcal{S}, \mathcal{O}^r, \mathcal{I}^r, \mathcal{G}^r \rangle$ with $\mathcal{O}^r \subseteq \mathcal{O}$, $\mathcal{I}^r \subseteq \mathcal{I}$, $\mathcal{G}^r \subseteq \mathcal{G}$)
$\rightarrow$ Every solution for $\mathcal{P}$ is one for $\mathcal{P}^r$
$\rightarrow$ The optimal solution cost of a relaxed problem is not greater than the optimal solution cost of the real problem

5.2.3 Greedy Search

$\rightarrow$ Uses a heuristic as evaluation function

- Not complete (only if finite space with repeated state checking)
- Time $\mathcal{O}(b^m)$
- Space $\mathcal{O}(b^m)$
- Not optimal

5.2.4 A* Search

$\rightarrow$ Evaluation function: $f(n) = g(n) + h(n)$
$\rightarrow$ $g(n)$ the path cost function, $h(n)$ the heuristic

- Complete if there are not infinitely many nodes with $f(n) \leq f(0)$
- Time: exponential in $relativeErrorInH \times lengthOfSolution$

- Space: exponential in $relativeErrorInH \times lengthOfSolution$
- **Optimal with admissible heuristic**

5.3 Local search

→ Options aren't searched systematically

→ operates on a single state (current state)

- Traveling Salesman:

  Find shortest trip through set of cities such that each city is only visited once

  → Start with any complete tour, perform pairwise exchanges

- n-queens problem

  Put n queens on a $n \times n$ board such that no two queens are in the same row, column, or diagonal

  → Move a queen to reduce number of conflicts

5.3.1 Hill-climbing (Gradient ascent/descent)

→ Starting anywhere + doing depth-first search with heuristic

→ only if solutions are dense and local maxima can be escaped

5.3.2 Simulated Annealing

→ Escape local maxima by allowing some "bad" moves, but gradually decrease their size and frequency

→ Shaking ping-pong ball on a bumpy surface

→ Ridges are ascending successions of maxima

5.3.3 Local Beam Search

$\rightarrow$ Keep k states instead of one

$\rightarrow$ Choose top k of all successors

5.3.4 Genetic Algorithms

$\rightarrow$ States encoded as strings with substrings as meaningful components

$\rightarrow$ Local beam search with random modifications of states, crossovers between pairs of states and optimizing fitness functions

6 Adversarial Search for Game Playing

$\rightarrow$ Discrete game states, finite number of game states, finite number of possible moves, fully observable game states, outcome of moves deterministic, two players, turn-taking, terminal states have utility, zero-sum utility (min tries to get opposite of max)

- Game state space: $\Theta := \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{I}, \mathcal{S}^T, u \rangle$

  States: $\mathcal{S} = \mathcal{S}^T \cup \mathcal{S}^{Max} \cup \mathcal{S}^{Min}$

  Actions: $\mathcal{A}$

  Transition relation: $\mathcal{T}$

  Initial state: $\mathcal{I}$

  Terminal states: $\mathcal{S}^T$

  Utility function: $u : \mathcal{S}^T \rightarrow \mathbb{R}$

- Position = State, End State = Terminal State, Move = Action

- Strategy: $\sigma^X : \mathcal{S}^X \rightarrow \mathcal{A}^X \quad$ with $X \in \{Max, Min\}$

  $\rightarrow$ optimal if it yields best possible utility for X assuming perfect opponent play

Game descriptions:

- Explicit
- Blackbox

  → with human knowledge
- Declarative (General Game Playing)

  → only given the rules

6.1 Minimax Search

→ We are Max and our opponent is Min

→ Max tries to maximize $u(s)$

→ Min tries to minimize $u(s)$

→ Computation alternates between Min and Max

1. Depth-first search in game tree with Max in the root
2. Apply utility function to terminal positions
3. Bottom-up compute $u(n)$:
   - Max's turn: $u(n)$ = maximum of utilities of n's successor nodes
   - Min's turn: $u(n)$ = minimum of utilities of n's successor nodes
4. Choose move that leads to successor node with maximal utility

!Infeasible → use search depth limits and evaluation functions!

6.2 Evaluation Functions

$\rightarrow$ $f(s) \rightarrow$ estimate of $u(s)$

$\rightarrow$ If cut-off states are terminal states use u instead of f

- Weighted linear function: $f = w_1 f_1 + w_2 f_2 + ... + w_n f_n$ with w_i weights and f_i features

  w_i can be learned automatically

  f_i have to be assigned by humans

  f_i in chess: Material, Mobility, King Safety...

6.3 Quiescence

$\rightarrow$ Iterative deepening with dynamically adapted depth

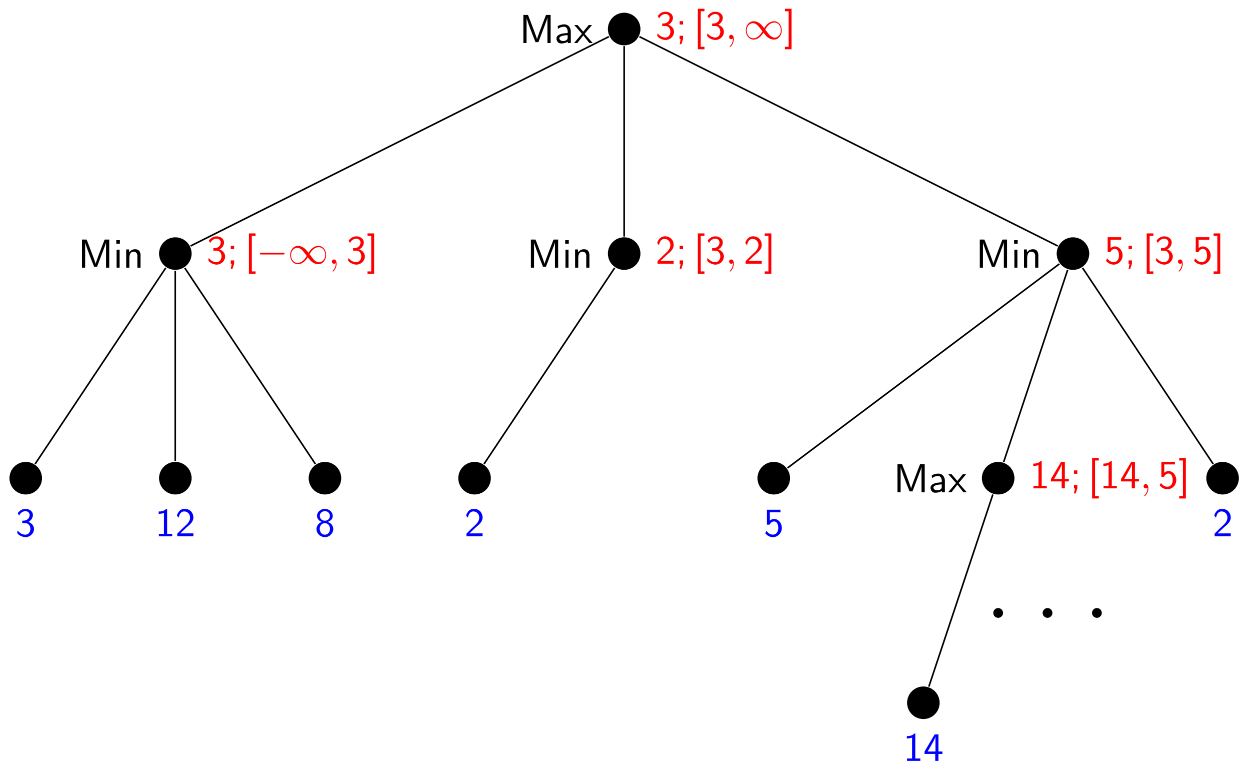
$\rightarrow$ Search more in positions where value f changes a lot in neighbouring positions

6.4 Alpha-Beta Search

- α = the **highest Max-node utility** that search has encountered on its path from the root to n
- β = the **lowest Min-node utility** that search has encountered on its path from the root to n

– $\alpha-$Pruning: In a **Min node** n, if one successor already has utility $\leq \alpha$, then stop considering n

– $\beta-$Pruning: In a **Max node** n, if one successor already has utility $\geq \beta$, then stop considering n



$\rightarrow$ Don't look at nodes, where $\alpha > \beta$

$\rightarrow$ If best moves are always chosen first: $\mathcal{O}(b^{\frac{d}{2}})$
b: Branching factor, d: depth limit

6.5 Monte-Carlo Tree Search

1. Try random paths from current state s
2. Take for each child of s the average of found utilities
3. Decide for child with biggest average

$\rightarrow$ Better in runtime and memory

$\rightarrow$ Needs good guidance for selecting and sampling

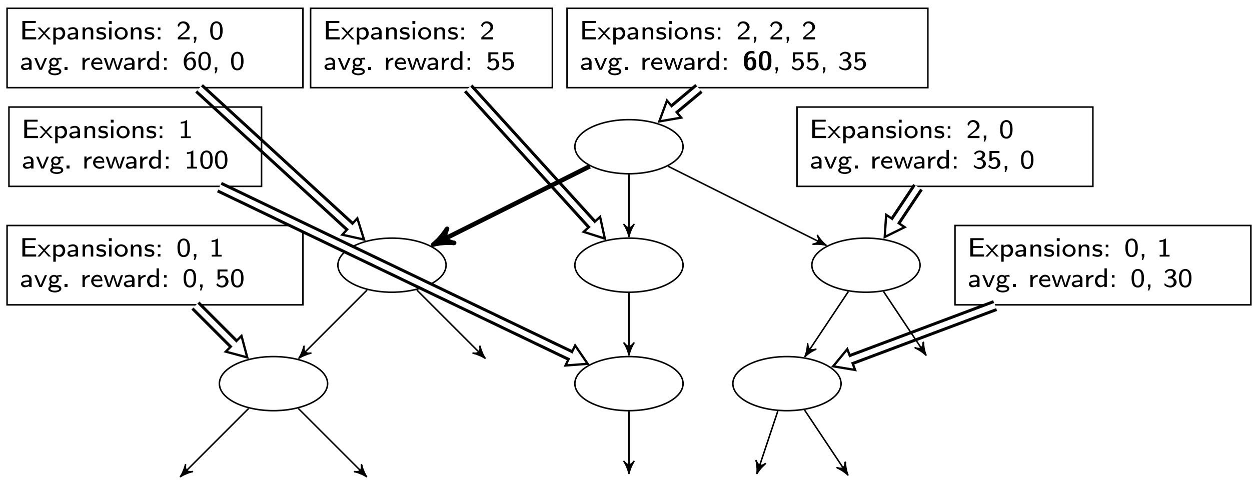
Sample-balancing:

- Exploitation: Prefer moves with high average
- Exploration: Prefer moves that have not been tried a lot

$\rightarrow$ Upper Confidence bounds applied to Trees (UCT) (formula defining balance)

With Tree building:

keep track of your average utilities also in children



7 Constraint Satisfaction Problems

Constraint Satisfaction problem:

- Search problem
- States:

  Variables: $V = \{X_1, ..., X_n\}$

  Domains: $\{D_v | v \in V\}$

- Goal Test:

  Constraints: Allowable combinations of values for subsets of variables

Complexity:

∗ n discrete variables

  – Finite domains with size d: $\mathcal{O}(d^n)$

  – Infinite domains

    With linear constraints solvable

    With nonlinear constraints undecidable

∗ Continuous variables

  – Linear constraints: solvable in poly time by linear programming

  – Nonlinear constraints: Not solvable

$\Rightarrow$ NP-complete to decide if solvable or not

$\rightarrow$ at most n^2 constraints, each of size at most d^2 $\rightarrow \mathcal{O}(n^2k^2)$

Types of Constraints:

- Unary: only one variable involved
- Binary: pairs of variables involved
- Higher-order: more variables involved
- Preferences: constraints with costs (when broken)

Constraint network $\langle V, D, C\rangle$

- Finite set of variables: $V = \{X_1, ..., X_n\}$
- Set of variables' domains: $D = \{D_v | v \in V\}$
- $C = \{C_{uv} | u, v \in V \text{ and } u \neq v\}$, where a binary constraint C_{uv} is a relation ($C_{uv} \subseteq D_u \times D_v$) and $C_{uv} = C_{vu}$

$\rightarrow$ Binary Constraint Satisfaction Problems can be reformulated as constrained networks

Partial assignment:
partial function $a : V \rightarrow \bigcup_{u \in V} D_u$ with $a(v) \in D_v$ for all $v \in$ *dom*(V) mit *dom*(x) = Wertebereich von x

Inconsistency:
A partial assignment is called inconsistent, if there are variables $u, v \in$ *dom*(a) and $C_{uv} \in C$, but $(a(u), a(v)) \notin C_{uv}$

$\rightarrow$ empty assignment is consistent

Extension:
partial assignment f extends partial assignment g, if $dom(g) \subseteq dom(f)$ and $f|_{dom(g)} = g$

Solution:
γ is a constraint satisfaction problem, then a consistent total assignment is a solution and γ is solvable.

7.1 Waltz Algorithm

- Problem: Interpret line drawings of solid polyhedra. Are intersections concave or convex?
- Assumptions:

  No shadows, cracks

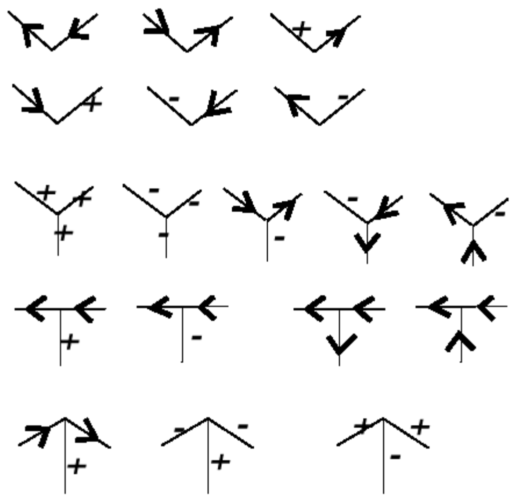
  only three-faced vertices

  no junctions change with small movements of the eye
- Each line is either

  > with right hand of arrow = space,
  with left hand of arrow = solid

  or + interior convex edge

  or − interior concave edge
- Constraints:



7.2 CSP as Search

- Initial State: empty assignment
- Successor function: assign value to unassigned variable that produces no conflict
- fail if no legal assignments

- Goal test: current assignment is complete

→ Same fo all CSPs

→ Every solution is at depth n

Backtracking search
→ Depth-first search for CSPs with single-variable assignments
→ Reihenfolge der Belegung muss egal sein

Heuristic: Minimum Remaining Values
→ Choose most constrained *variable* first

Degree Heuristic
→ Choose *variable* with most constraints on remaining variables first

Commonly used strategy combination:
→ From set of most constrained variables choose the most constraining one

Least Constraining Value Heuristic
→ Given a variable, choose the least constraining *value*

8 Constraint Propagation

8.1 Inference

→ find additional constraints, that follow from the already known constraints

→ Replace γ by an equivalent and strictly tighter constraint network γ'

Equivalent Constraint Networks
γ and γ' (have the same set of variables) are equivalent ($\equiv$), if they have the same solutions.

Tightness
γ' is tighter ($\sqsubseteq$) than γ, if:

- For all $v \in V : D'_v \subseteq D_v$
- For all $u \neq v \in V :$ either $C_{uv} \notin C$ or $C'_{uv} \subseteq C_{uv}$

γ' is strictly tighter ($\sqsubset$) than γ, if at least one of these inclusions is strict

Inference
$\equiv + \sqsubset =$ Inference
$\gamma' \equiv \gamma$ and $\gamma' \sqsubset \gamma$ then γ' has the same solutions as γ, but fewer consistent partial assignments
$\rightarrow$ γ' is a better encoding of the underlying problem

8.1.1 Backtracking with Inference

- Inference at every recursive call of backtracking
- Search vs. Inference: The more complex the inference, the smaller the number of search nodes, but the larger the runtime needed at each node.
- Encode partial assignment as unary constraints (i.e., for $a(v) = d$, set the unary constraint $D_v = \{d\}$), so that inference reasons about the network restricted to the commitments already made.

8.2 Forward Checking

$\rightarrow$ Inference

```
function $ForwardChecking(\gamma, a)$ returns modified $\gamma$
    for each $v$ where $a(v) = d'$ is defined do
        for each $u$ where $a(u)$ is undefined and $C_{uv} \in C$ do
            $D_u = \{d \in D_u | (d, d') \in C_{uv}\}$
    return $\gamma$
```

→ Forward Checking is sound: Tightening does not rule out solutions

8.3 Arc Consistency

→ Inference

- A variable $u \in V$ is **arc consistent** relative to another variable $v \in V$ if either $C_{uv} \notin C$, or for every value $d \in D_u$ there exists a value $d' \in D_v$ such that $(d, d') \in C_{uv}$.

  → arc consistency is directed/asymmetric

- The network γ is arc consistent if every variable $u \in V$ is arc consistent relative to every other variable $v \in V$.

→ Arc Consistency is sound: Guarantees to deliver an equivalent network

→ Arc Consistency subsumes forward checking:

$$AC(\gamma) \sqsubseteq ForwardChecking(\gamma)$$

8.3.1 Arc Consistency for one pair of variables

```
function Revise ($\gamma, u, v$) returns modified $\gamma$
    for each $d \in D_u$ do
        if there is no $d' \in D_v$ with $(d, d') \in C_{uv}$ then $D_i := D_u \setminus \{d\}$
    return $\gamma$
```

$\rightarrow$ $\mathcal{O}(k^2)$ with k the maximal domain size

8.3.2 AC-1

```
function $AC-1(\gamma)$ returns modified $\gamma$
    repeat
        $changesMade := False$
        for each constraint $C_{uv}$ do
            $Revise(\gamma, u, v)$ /* if $D_u$ reduces, set $changesMade := True$ */
            $Revise(\gamma, v, u)$ /* if $D_v$ reduces, set $changesMade := True$ */
    until $changesMade = False$
    return $\gamma$
```

$\rightarrow$ $\mathcal{O}(mk^2nk)$ with n variables, m constraints, k maximal domain size

$\rightarrow$ Redundant computations

8.3.3 AC-3

```
function $AC-3(\gamma)$ returns modified $\gamma$
    $M := \emptyset$
    for each constraint $C_{uv} \in C$ do
        $M := M \cup \{(u,v),(v,u)\}$
    while $M \neq \emptyset$ do
        remove any element $(u,v)$ from $M$
        $Revise(\gamma, u, v)$
        if $D_u$ has changed in the call to Revise then
            for each constraint $C_{wu} \in C$ where $w \neq v$ do
                $M := M \cup \{(w,u)\}$
    return $\gamma$
```

$\rightarrow$ $\mathcal{O}(mk^3)$ with m constraints, k maximal domain size

8.4 Decomposition

$\rightarrow$ Often, we can exploit the structure of a network to decompose it into smaller parts that are easier to solve

8.5 Constraint Graphs

$\rightarrow$ Decomposition

8.5.1 Disconnected Constraint Graphs

Let $\gamma = \langle V, D, C \rangle$ be a constraint network. Let a_i be a solution to each connected component V_i of the constraint graph of γ. Then $a := \bigcup_i a_i$ is a solution to γ.

$\rightarrow$ Reduction of worst-case

8.5.2 Acyclic Constraint Graphs

Let $\gamma = \langle V, D, C \rangle$ be a constraint network with n variables and maximal domain size k, whose constraint graph is acyclic. Then we can find a solution for γ, or prove γ to be inconsistent, in time $\mathcal{O}(nk^2)$.

```
$AcyclicCG(\gamma)$
  1. Obtain a directed tree from $\gamma$'s constraint graph, picking an
     arbitrary variable $v$ as the root, and directing arcs outwards
  2. Order the variables topologically, i.e., such that each vertex
     is ordered before its children; denote that order by $v_1, ..., v_n$
  3. **for** $i := n, n-1, ..., 2$ **do**
         $Revise(\gamma, v_{parent(i)}, v_i)$
         **if** $D_{v_{parent(i)}} = \emptyset$ **then return** "inconsistent"
  $\rightarrow$ Every variable is arc consistent relative to its children
  4. Run $BacktrackingWithInference$ with forward checking,
     using the variable order $v_1, ..., v_n$
```

$\Rightarrow$ This algorithm will find a solution without ever having to backtrack!

8.6 Cutset Conditioning

$\rightarrow$ Decomposition

1. Recursive call of backtracking on a, the sub-graph of the constraint graph induced by $\{v \in V | a(v) \text{ is undefined}\}$ is acyclic.

$\rightarrow$ use $AcyclicCG()$ for sub-graph

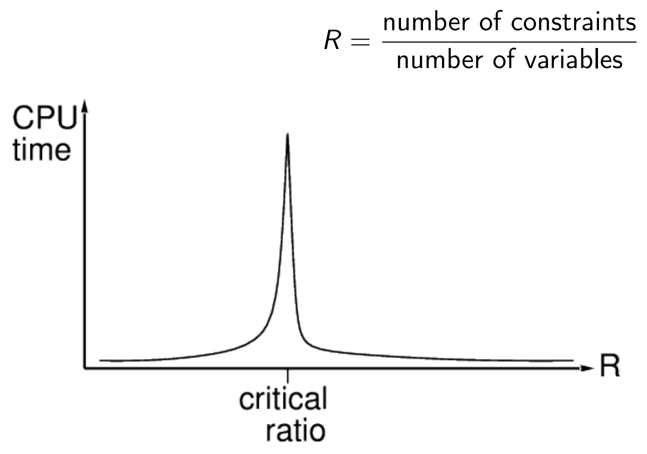
2. Choose the variable order so that removing the first d variables renders the constraint graph acyclic

$\Rightarrow$ Runtime exponential in # of variables in the sub-graph not the whole graph

$\Rightarrow$ Finding optimal cutsets is NP-hard

8.7 Constraint Propagation with Local Search

- Allow states with unsatisfied constraint operators to reassign variable values
- Variable selection: randomly select any conflicted variable
- Value selection: by min-conflicts heuristic: choose value that violates the fewest constraints



9 Knowledge and Inference

A Logic is decidable, when its satisfiability problem can be decided in finite time

9.1 Propositional Reasoning

Representing Knowledge

- Syntax

  What are legal statements (formulas) $\mathbf{A}$ in the logic

- Semantics

  Which formulas $\mathbf{A}$ are true under which assignment ϕ

  written $\phi \models \mathbf{A}$

Reasoning about Knowledge

- Entailment

  Which $\mathbf{B}$ are entailed by $\mathbf{A}$

  written $\mathbf{A} \models \mathbf{B}$

  meaning for all ϕ with $\phi \models \mathbf{A}$, we have $\phi \models \mathbf{B}$

- Deduction

  Which statements $\mathbf{B}$ can be derived from $\mathbf{A}$
  using a set $\mathcal{C}$ of inference rules (a calculus)
  written $\mathbf{A} \vdash_{\mathcal{C}} \mathbf{B}$

  Properties of Deduction:

  – Calculus soundness
    - → whenever $\mathbf{A} \vdash_{\mathcal{C}} \mathbf{B}$, we also have $\mathbf{A} \models \mathbf{B}$
    - → I don't pretend to know, if I don't know
    - → A calculus is correct if any derivable(provable) formula is also a valid formula

  – Calculus completeness
    - → whenever $\mathbf{A} \models \mathbf{B}$, we also have $\mathbf{A} \vdash_{\mathcal{C}} \mathbf{B}$
    - → When I have enough knowledge, I can also deduce it
    - → A calculus is complete if any valid formula can also be derived(proven)

9.2 Propositional Logic (PL^0)

9.2.1 Syntax

= atomic propositions

- Propositional variables: $\mathcal{V}_o$
- Connectives: $\Sigma_o := \{T, F, \neg, \wedge, \vee, \implies, \iff, ...\}$
- Well-formed propositional formulas: $wff_o(\mathcal{V}_o)$

  Negation $\neg\mathbf{A}$

Conjunction $\mathbf{A} \wedge \mathbf{B}$

Disjunction $\mathbf{A} \vee \mathbf{B}$

Implication $\mathbf{A} \implies \mathbf{B}$

Equivalences/Biimplication $\mathbf{A} \iff \mathbf{B}$

$\rightarrow$ with $\mathbf{A}, \mathbf{B} \in wff_o(\mathcal{V}_o)$

- propositional formulae without connectives are called atomic (or atoms) and complex otherwise

9.2.2 Semantics

= Assign value to every proposition

- Model $\mathcal{M} := \langle \mathcal{D}_o, \mathcal{I} \rangle$
  - Universe $\mathcal{D}_o = \{T, F\}$
  - Interpretation $\mathcal{I}$ assigns values to connectives

    $\mathcal{I}(\neg) = \mathcal{D}_o \rightarrow \mathcal{D}_o; \quad T \mapsto F, F \mapsto T$

    $\mathcal{I}(\wedge) = \mathcal{D}_o \times \mathcal{D}_o \rightarrow \mathcal{D}_o; \; \langle \alpha, \beta \rangle \mapsto T, \text{ if } \alpha = \beta = T$

    $\rightarrow$ other connectives can be represented by these two
- Variable assignment $\phi : \mathcal{V}_o \rightarrow \mathcal{D}_o$

  $\rightarrow$ assigns values to propositional variables
- Value function $\mathcal{I}_\phi : wff_o(\mathcal{V}_o) \rightarrow \mathcal{D}_o$ or $[\![\mathbf{A}]\!]_\phi^\mathcal{M}$

  $\rightarrow$ assigns values to formulae

  $\mathcal{I}_\phi(P) = \phi(P)$

  $\mathcal{I}_\phi(\neg\mathbf{A}) = \mathcal{I}(\neg)(\mathcal{I}_\phi(\mathbf{A}))$

  $\mathcal{I}_\phi(\mathbf{A} \wedge \mathbf{B}) = \mathcal{I}(\wedge)(\mathcal{I}_\phi(\mathbf{A}), \mathcal{I}_\phi(\mathbf{B}))$

Definitions:

– $\mathbf{A}$ is true under ϕ (ϕ satisfies $\mathbf{A}$) in $\mathcal{M}$
  if $\mathcal{I}_\phi(\mathbf{A}) = T$

– $\mathbf{A}$ is false under ϕ (ϕ falsifies $\mathbf{A}$) in $\mathcal{M}$
  if $\mathcal{I}_\phi(\mathbf{A}) = F$

– $\mathbf{A}$ is satisfiable in $\mathcal{M}$
  if $\mathcal{I}_\phi(\mathbf{A}) = T$ for some assignments ϕ

– $\mathbf{A}$ is valid in $\mathcal{M}$
  if $\mathcal{M} \models^\phi \mathbf{A}$ for all assignments ϕ

– $\mathbf{A}$ is falsifiable in $\mathcal{M}$
  if $\mathcal{I}_\phi(\mathbf{A}) = F$ for some assignments ϕ

– $\mathbf{A}$ is unsatisfiable in $\mathcal{M}$
  if $\mathcal{I}_\phi(\mathbf{A}) = F$ for all assignments ϕ

– $\mathbf{A}$ entails $\mathbf{B}$ ($\mathbf{A} \models \mathbf{B}$)
  if $\mathcal{I}_\phi(\mathbf{B}) = T$ for all ϕ with $\mathcal{I}_\phi(\mathbf{A}) = T$

9.3 Formal Systems

A **logical system** is a triple $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ where $\mathcal{L}$ is a formal language, $\mathcal{K}$ is a set and $\models \subseteq := \mathcal{K} \times \mathcal{L}$. Members of $\mathcal{L}$ are called formulae of $\mathcal{S}$, members of $\mathcal{K}$ models for $\mathcal{S}$, and $\models$ the satisfaction relation.

Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system, then we call a relation $\vdash \subseteq \mathcal{P}(\mathcal{L}) \times \mathcal{L}$ a **derivation relation** for $\mathcal{S}$, if it:

- is proof-reflexive $\mathcal{H} \vdash \mathbf{A}$, if $\mathbf{A} \in \mathcal{A}$
- is proof-transitive $\mathcal{H} \vdash \mathbf{A}$ and $\mathcal{H}' \cup \{\mathbf{A}\} \vdash \mathcal{B}$, then $\mathcal{H} \cup \mathcal{H}' \vdash \mathcal{B}$

- monotonic $\mathcal{H} \vdash \mathbf{A}$ and $\mathcal{H} \subseteq \mathcal{H}'$ imply $\mathcal{H}' \vdash \mathbf{A}$

We call $\langle \mathcal{L}, \mathcal{K}, \models, \vdash \rangle$ a **formal system**, iff $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ is a logical system, and $\vdash$ a derivation relation for $\mathcal{S}$

Let $\mathcal{L}$ be a formal language, then an **inference rule** over $\mathcal{L}$

$$\frac{\mathbf{A}_1 \quad \cdots \quad \mathbf{A}_n}{\mathbf{C}}\mathcal{N}$$

where A_1, ..., A_n and C are formula schemata for $\mathcal{L}$ and $\mathcal{N}$ is a name. The A_i are called assumptions, and C is called conclusion

An inference rule without assumptions is called an **axiom** (schema).

Let $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ be a logical system, then we call a set $\mathcal{C}$ of inference rules over $\mathcal{L}$ a **calculus** for $\mathcal{S}$

We call $\langle \mathcal{L}, \mathcal{K}, \models, \mathcal{C} \rangle$ a **formal system**, iff $\mathcal{S} := \langle \mathcal{L}, \mathcal{K}, \models \rangle$ is a logical system, and $\mathcal{C}$ a calculus for $\mathcal{S}$.

A derivation $\emptyset \vdash_{\mathcal{C}} \mathbf{A}$ is called a **proof** of $\mathbf{A}$ and if one exists (write $\vdash_{\mathcal{C}} \mathbf{A}$) then $\mathbf{A}$ is called a $\mathcal{C}$-theorem.

9.4 Propositional Natural Deduction Calculus ($\mathcal{ND}^0$)

$$\frac{\mathbf{A} \quad \mathbf{B}}{\mathbf{A} \wedge \mathbf{B}} \wedge I \qquad \frac{\mathbf{A} \wedge \mathbf{B}}{\mathbf{A}} \wedge E_l \quad \frac{\mathbf{A} \wedge \mathbf{B}}{\mathbf{B}} \wedge E_r \qquad \frac{}{\mathbf{A} \vee \neg \mathbf{A}} \mathrm{TND}$$

$$\frac{\begin{array}{c}[\mathbf{A}]^1 \\ \hline\hline \mathbf{B}\end{array}}{\mathbf{A} \Rightarrow \mathbf{B}} \Rightarrow I^1 \qquad \frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}} \Rightarrow E$$

used only in classical logic (otherwise constructive/intuitionistic

$$\frac{\mathbf{A}}{\mathbf{A}\vee\mathbf{B}}\vee I_l \qquad \frac{\mathbf{B}}{\mathbf{A}\vee\mathbf{B}}\vee I_r \qquad \frac{\mathbf{A}\vee\mathbf{B} \quad \begin{matrix}[\mathbf{A}]^1\\ \vdots \\ \mathbf{C}\end{matrix} \quad \begin{matrix}[\mathbf{B}]^1\\ \vdots \\ \mathbf{C}\end{matrix}}{\mathbf{C}}\vee E^1$$

$$\frac{\begin{matrix}[\mathbf{A}]^1\\ \vdots \\ F\end{matrix}}{\neg\mathbf{A}}\neg I^1 \qquad \frac{\neg\neg\mathbf{A}}{\mathbf{A}}\neg E$$

$$\frac{\neg\mathbf{A} \quad \mathbf{A}}{F}FI \qquad \frac{F}{\mathbf{A}}FE$$

$\mathcal{H}, \mathbf{A} \vdash_{\mathcal{ND}^0} \mathbf{B}$, iff $\mathcal{H} \vdash_{\mathcal{ND}^0} \mathbf{A} \implies \mathbf{B}$

9.5 Machine-Oriented Calculi for Propositional Logic

Unsatisfiability Theorem: Iff $\mathcal{H} \cup \{\neg\mathbf{A}\}$, $\mathcal{H} \models \mathbf{A}$ is unsatisfiable

- Conjunctive Normal Form (CNF): $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} I_{i,j}$
- Disjunctive Normal Form (DNF): $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} I_{i,j}$

9.5.1 Analytic Tableaux

We call a formula **atomic**, or an atom, iff it does not contain connectives. We call a formula **complex**, iff it is not atomic.
We call a pair $\mathbf{A}^{\alpha}$ labeled formula, if $\alpha \in \{T, F\}$. A labeled atom is called **literal**.

$\Rightarrow$ Instead of showing $\emptyset \vdash Th$, show $\neg Th \vdash \perp$

- formula is analyzed in a tree to determine satisfiability

$\rightarrow$ Satisfiable, iff there are open branches

- Use rules exhaustively as long as they contribute new material:

$$\frac{\mathbf{A} \wedge \mathbf{B}^{\mathsf{T}}}{\begin{array}{c}\mathbf{A}^{\mathsf{T}}\\ \mathbf{B}^{\mathsf{T}}\end{array}}\mathcal{T}_0\wedge \qquad \frac{\mathbf{A} \wedge \mathbf{B}^{\mathsf{F}}}{\mathbf{A}^{\mathsf{F}} \;\big|\; \mathbf{B}^{\mathsf{F}}}\mathcal{T}_0\vee \qquad \frac{\neg\,\mathbf{A}^{\mathsf{T}}}{\mathbf{A}^{\mathsf{F}}}\mathcal{T}_0{}^{\mathsf{T}}_{\neg} \qquad \frac{\neg\,\mathbf{A}^{\mathsf{F}}}{\mathbf{A}^{\mathsf{T}}}\mathcal{T}_0{}^{\mathsf{F}}_{\neg} \qquad \frac{\begin{array}{c}\mathbf{A}^{\alpha}\\ \mathbf{A}^{\beta}\end{array} \quad \alpha \neq \beta}{\perp}\mathcal{T}_0\mathsf{cut}$$

$\rightarrow$ Call a tableau *saturated*, iff no rule applies, and a branch *closed*, iff it ends in $\perp$, else *open*

$\rightarrow$ A is a $\mathcal{T}_0$-theorem ($\vdash_{\mathcal{T}_0} \mathbf{A}$), iff there is a closed tableau with $\mathbf{A}^F$ at the root

$\rightarrow$ $\phi \subseteq wff_o(\mathcal{V}_o)$ *derives* $\mathbf{A}$ in $\mathcal{T}_o(\phi \vdash_{\mathcal{T}_o} \mathbf{A})$, iff there is a closed tableau starting with $\mathbf{A}^F$ and ϕ^T

$\rightarrow$ Terminating tableaux are a tableau calculus with the property that any of its derivations terminates after finitely many steps

- Derived rules:

$$\frac{\mathbf{A} \Rightarrow \mathbf{B}^{\mathsf{T}}}{\mathbf{A}^{\mathsf{F}} \;\big|\; \mathbf{B}^{\mathsf{T}}} \qquad \frac{\mathbf{A} \Rightarrow \mathbf{B}^{\mathsf{F}}}{\begin{array}{c}\mathbf{A}^{\mathsf{T}}\\ \mathbf{B}^{\mathsf{F}}\end{array}} \qquad \frac{\begin{array}{c}\mathbf{A}^{\mathsf{T}}\\ \mathbf{A} \Rightarrow \mathbf{B}^{\mathsf{T}}\end{array}}{\mathbf{B}^{\mathsf{T}}}$$

$$\frac{\mathbf{A} \vee \mathbf{B}^{\mathsf{T}}}{\mathbf{A}^{\mathsf{T}} \;\big|\; \mathbf{B}^{\mathsf{T}}} \qquad \frac{\mathbf{A} \vee \mathbf{B}^{\mathsf{F}}}{\begin{array}{c}\mathbf{A}^{\mathsf{F}}\\ \mathbf{B}^{\mathsf{F}}\end{array}} \qquad \frac{\mathbf{A} \Leftrightarrow \mathbf{B}^{\mathsf{T}}}{\begin{array}{c|c}\mathbf{A}^{\mathsf{T}} & \mathbf{A}^{\mathsf{F}}\\ \mathbf{B}^{\mathsf{T}} & \mathbf{B}^{\mathsf{F}}\end{array}} \qquad \frac{\mathbf{A} \Leftrightarrow \mathbf{B}^{\mathsf{F}}}{\begin{array}{c|c}\mathbf{A}^{\mathsf{T}} & \mathbf{A}^{\mathsf{F}}\\ \mathbf{B}^{\mathsf{F}} & \mathbf{B}^{\mathsf{T}}\end{array}}$$

9.5.2 Resolution

$\rightarrow$ The proof goal is transformed in CNF and then (dis)proved through resolution refutation

$\rightarrow$ $\{A, \neg B\}, \{\neg A, B\} \Rightarrow \{A, \neg A\}, \{\neg B, B\}$

- Resolution Calculus:

$$\frac{P^{\mathsf{T}} \vee \mathbf{A} \quad P^{\mathsf{F}} \vee \mathbf{B}}{\mathbf{A} \vee \mathbf{B}}$$

- Resolution Refutation:

  $\mathcal{D} : S \vdash_{\mathcal{R}} \Box$ with derivation $\mathcal{R}$ and clause set S

- Resolution Proof:

  We call a resolution refutation of $CNF^0(A^F)$ a resolution proof for $A \in wff_o(\mathcal{V}_o)$

- Clause = disjunction of literals ($\Box$ = empty disjunction)

- Clause Normal Transformation:

$$\frac{\mathbf{C} \vee (\mathbf{A} \vee \mathbf{B})^{\mathsf{T}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{T}} \vee \mathbf{B}^{\mathsf{T}}} \quad \frac{\mathbf{C} \vee (\mathbf{A} \vee \mathbf{B})^{\mathsf{F}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{F}}; \mathbf{C} \vee \mathbf{B}^{\mathsf{F}}} \qquad \frac{\mathbf{C} \vee \neg \mathbf{A}^{\mathsf{T}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{F}}} \quad \frac{\mathbf{C} \vee \neg \mathbf{A}^{\mathsf{F}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{T}}}$$

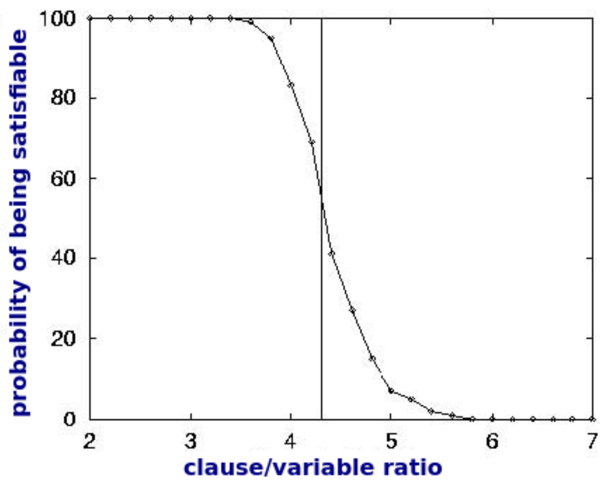
- Derived rules of inference:

  a rule $\frac{\mathbf{A}_1 \quad \ldots \quad \mathbf{A}_n}{\mathbf{C}}$ is a derived inference rule in the calculus $\mathcal{C}$, iff there is a $\mathcal{C}$-proof of $A_1, ..., A_n \vdash C$

$$\frac{\mathbf{C} \vee (\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{T}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{F}} \vee \mathbf{B}^{\mathsf{T}}} \quad \frac{\mathbf{C} \vee (\mathbf{A} \Rightarrow \mathbf{B})^{\mathsf{F}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{T}}; \mathbf{C} \vee \mathbf{B}^{\mathsf{F}}} \qquad \frac{\mathbf{C} \vee \mathbf{A} \wedge \mathbf{B}^{\mathsf{T}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{T}}; \mathbf{C} \vee \mathbf{B}^{\mathsf{T}}} \quad \frac{\mathbf{C} \vee \mathbf{A} \wedge \mathbf{B}^{\mathsf{F}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{F}} \vee \mathbf{B}^{\mathsf{F}}}$$

9.6 SAT Solver

SAT solvers decide satisfiability of CNF (Conjunctive Normal Form) formulas
The SAT problem consists in deciding whether a propositional formula is satisfiable. The problem's worst case complexity is NP-complete



Around a clause-to-variable ratio of (4.3) the SAT-problem becomes intractable. This is known as phase transition.

Clause Normal Form = A propositional logic formula consisting of conjunctions of disjunctions of literals

Any SAT problem can be viewed as a CSP-problem and any CSP-problem can be transformed into a SAT-problem in polynomial time.

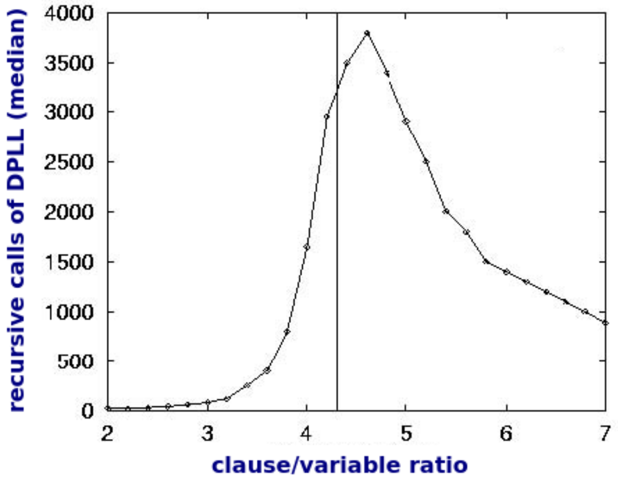
9.6.1 DPLL

DPLL = backtracking with inference performed by unit propagation (UP), which iteratively instantiates unit clauses and simplifies the formula

The DPLL algorithm uses the unit propagation rule and the split rule.

DPLL with clause learning is equivalent to resolution

$\rightarrow$ can be exponentially long



1. solange unit clause enthalten, wende unit propagation an
2. wenn $\Box$ erhalten $\rightarrow$ unsatisfiable
3. nehme eine random Proposition P und versuche DPLL mit einmal P = True und einmal P = False (gehe zu 1)
4. wenn für P^F und P^T kein unsatisfiable gefunden, dann haben wir eine partielle interpretation gefunden (alle bereits gesetzten propositions müssen diese Belegung haben, bei den nicht gesetzten ist die Belegung egal)

Unit Resolution:

$$\frac{C \vee P^{\mathrm{F}} \quad P^{\mathrm{T}}}{C}$$

Unit propagation = Resolution restricted to the case where one of the parent clauses is unit

9.7 First Order Predicate Logic PL^1

Syntax:

- Individual variables $\mathcal{V}_\iota$
  - Truth values o
  - Individuals ι
- Connectives Σ^o

  $\rightarrow$ on truth values
- Function constants $\Sigma_k^f = \{f, g, h, ...\}$

  $\rightarrow$ on individuals
- Predicate constants $\Sigma_k^p = \{p, q, r, ...\}$
- Skolem constants $\Sigma_k^{sk} = \{f_1^k, f_2^k, ...\}$
- $\Sigma_\iota = \Sigma^f \cup \Sigma^p \cup \Sigma^{sk}$
- $\Sigma = \Sigma_\iota \cup \Sigma^o$
- Formulae
  - Terms

    $\rightarrow$ denote individuals
  - Propositions

    $\rightarrow$ denote truth values
- Fixed truth values Universe $\mathcal{D}_o = \{T, F\}$
- Individuals Universe $\mathcal{D}_\iota \neq \emptyset$
- Model $\mathcal{M} = \langle \mathcal{D}_\iota, \mathcal{I} \rangle$

Semantics:

- As for propositional logic +

- $\mathcal{I}_\phi(f(A_1, ..., A_k)) = \mathcal{I}(f)(\mathcal{I}_\phi(A_1), ..., \mathcal{I}_\phi(A_k))$
- $\mathcal{I}_\phi(p(A^1, ..., A^k)) = \top$, iff $\langle \mathcal{I}_\phi(A^1), ..., \mathcal{I}_\phi(A^k) \rangle \in \mathcal{I}(p)$
- $\mathcal{I}_\phi(\forall X.A) = \top$, iff $\mathcal{I}_{\phi,[a/X]}(A) = \top$ for all $a \in \mathcal{D}_\iota$

$\rightarrow$ variable capture = An (unsound) operation that turns a free variable into a bound variable

A variable X is bound in a formula if and only if it occurs in the scope of either a universal or an existential quantifier binding X.

9.7.1 Natural Deduction $\mathcal{ND}^1$

$\mathcal{ND}^0$ +

$$\frac{\mathbf{A}}{\forall X.\mathbf{A}}\forall I^* \qquad \frac{\forall X.\mathbf{A}}{[\mathbf{B}/X](\mathbf{A})}\forall E$$

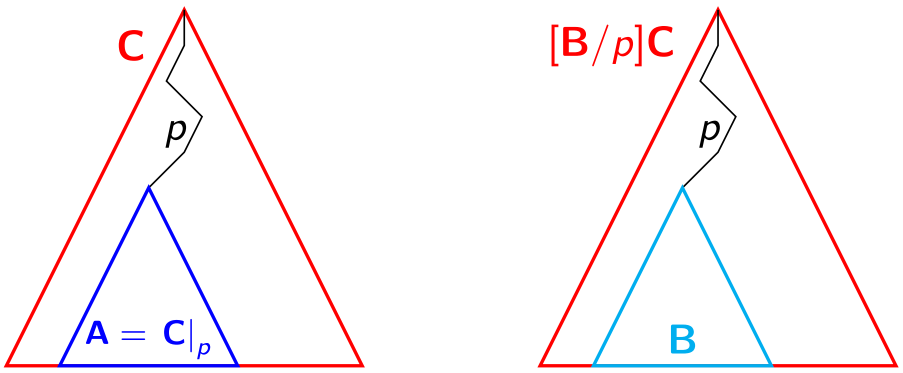
$$\frac{[\mathbf{B}/X](\mathbf{A})}{\exists X.\mathbf{A}}\exists I \qquad \frac{\exists X.\mathbf{A} \qquad \begin{array}{c}[[c/X](\mathbf{A})]^1\\ \vdots\\ \mathbf{C}\end{array}}{\mathbf{C}}\exists E^1$$

$*$ means that A does not depend on any hypothesis in which X is free.

$$\frac{}{\mathbf{A} \Leftrightarrow \mathbf{A}}\Leftrightarrow I \qquad \frac{\mathbf{A} \Leftrightarrow \mathbf{B} \quad \mathbf{C}\,[\mathbf{A}]_p}{[\mathbf{B}/p]\mathbf{C}}\Leftrightarrow =E$$

9.7.2 First-Order Logic with Equality

$$\frac{}{\mathbf{A} = \mathbf{A}} {=}I \qquad \frac{\mathbf{A} = \mathbf{B} \quad \mathbf{C}\,[\mathbf{A}]_p}{[\mathbf{B}/p]\mathbf{C}} {=}E$$



9.8 First Order Inference

9.8.1 Tableau

→ There is no terminating tableaux calculus for first order logic

$\mathcal{T}_0 +$

$$\frac{\forall X.\mathbf{A}^{\mathsf{T}} \quad \mathbf{C} \in cwff_\iota(\Sigma_\iota)}{[\mathbf{C}/X](\mathbf{A})^{\mathsf{T}}} \mathcal{T}_1{:}\forall \qquad \frac{\forall X.\mathbf{A}^{\mathsf{F}} \quad c \in (\Sigma_0^{sk} \backslash \mathcal{H})}{[c/X](\mathbf{A})^{\mathsf{F}}} \mathcal{T}_1{:}\exists$$

→ have to guess in $\mathcal{T}_1 : \forall$

⇒ Free Variable Tableau $\mathcal{T}_0^f$:

$$\frac{\forall X.\mathbf{A}^{\mathsf{T}} \quad Y\,new}{[Y/X](\mathbf{A})^{\mathsf{T}}} \mathcal{T}_1^f{:}\forall \qquad \frac{\forall X.\mathbf{A}^{\mathsf{F}} \quad \mathrm{free}(\forall X.\mathbf{A}) = \{X^1, \ldots, X^k\} \quad f \in \Sigma_k^{sk}}{[f(X^1, \ldots, X^k)/X](\mathbf{A})^{\mathsf{F}}} \mathcal{T}_1^f{:}\exists$$

$$\frac{\begin{array}{l}\mathbf{A}^{\alpha}\\ \mathbf{B}^{\beta}\end{array}\quad \alpha \neq \beta \quad \sigma(\mathbf{A}) = \sigma(\mathbf{B})}{\bot : \sigma}\mathcal{T}_1^f{:}\bot$$

9.8.2 Resolution

CNF:

$$\frac{(\forall X\,\mathbf{.}\,\mathbf{A})^{\mathsf{T}} \vee \mathbf{C} \quad Z \notin (\mathrm{free}(\mathbf{A}) \cup \mathrm{free}(\mathbf{C}))}{[Z/X](\mathbf{A})^{\mathsf{T}} \vee \mathbf{C}}$$

$$\frac{(\forall X\,\mathbf{.}\,\mathbf{A})^{\mathsf{F}} \vee \mathbf{C} \quad \{X_1, \ldots, X_k\} = \mathrm{free}(\forall X\,\mathbf{.}\,\mathbf{A})}{[f_n^k(X^1, \ldots, X^k)/X](\mathbf{A})^{\mathsf{F}} \vee \mathbf{C}}$$

Calculus:

$$\frac{\mathbf{A}^{\mathsf{T}} \vee \mathbf{C} \quad \mathbf{B}^{\mathsf{F}} \vee \mathbf{D} \quad \sigma = \mathbf{mgu}(\mathbf{A}, \mathbf{B})}{\sigma(\mathbf{C}) \vee \sigma(\mathbf{D})} \qquad \frac{\mathbf{A}^{\alpha} \vee \mathbf{B}^{\alpha} \vee \mathbf{C} \quad \sigma = \mathbf{mgu}(\mathbf{A}, \mathbf{B})}{\sigma(\mathbf{A}) \vee \sigma(\mathbf{C})}$$

9.8.3 Unification

$$\frac{\mathcal{E} \wedge f(\mathbf{A}^1, \ldots, \mathbf{A}^n) =^{?} f(\mathbf{B}^1, \ldots, \mathbf{B}^n)}{\mathcal{E} \wedge \mathbf{A}^1 =^{?} \mathbf{B}^1 \wedge \ldots \wedge \mathbf{A}^n =^{?} \mathbf{B}^n}\mathcal{U}\,\mathrm{dec} \qquad \frac{\mathcal{E} \wedge \mathbf{A} =^{?} \mathbf{A}}{\mathcal{E}}\mathcal{U}\,\mathrm{triv}$$

$$\frac{\mathcal{E} \wedge X =^{?} \mathbf{A} \quad X \notin \mathrm{free}(\mathbf{A}) \quad X \in \mathrm{free}(\mathcal{E})}{[\mathbf{A}/X](\mathcal{E}) \wedge X =^{?} \mathbf{A}}\mathcal{U}\,\mathrm{elim}$$

$\rightarrow$

- correct
- complete

- confluent → order of derivations does not matter

9.9 Logic Programming as Resolution Theorem Proving

- Deduction

$$\frac{\mathit{rains} \Rightarrow \mathit{wet\_street} \quad \mathit{rains}}{\mathit{wet\_street}}\,D$$

- Abduction

$$\frac{\mathit{rains} \Rightarrow \mathit{wet\_street} \quad \mathit{wet\_street}}{\mathit{rains}}\,A$$

- Learning rules

$$\frac{\mathit{wet\_street} \quad \mathit{rains}}{\mathit{rains} \Rightarrow \mathit{wet\_street}}\,I$$

Semantic Web = project of extending the internet by annotating content on the internet using logic to render it machine-readable

10 Planning & Acting

→ Write one program that can solve all classical search problems

Planning Language:

- States
- Initial State I
- Goal Condition G

- Actions A
  - Preconditions
  - Effects
- Solution → Plan (Sequence of actions)

10.1 STRIPS

→ Stanford Research Institute Problem Solver

→ the simplest possible (reasonably expressive) logics-based planning language

→ only Boolean variables

$\Pi = \langle P, A, I, G \rangle$

- P finite set of facts/Propositions
- A finite set of actions $A = \langle pre_a, add_a, del_a \rangle$
  - Preconditions pre_a
  - Add List add_a
  - Delete List del_a

    $add_a \cap del_a = \emptyset$
- Initial State $I \subseteq P$
- Goal $G \subseteq P$

10.2 PDDL

→ Planning Domain Description Language

- Domain file

```
(define (domain blocksworld)
(:predicates (clear ?x) (holding ?x) (on ?x ?y)
               (on−table ?x) (arm−empty))
(:action stack
  :parameters (?x ?y)
  :precondition (and (clear ?y) (holding ?x))
  :effect (and (arm−empty) (on ?x ?y)
                  (not (clear ?y)) (not (holding ?x)))
)
...
```

- Problem file

```
(define (problem bw−abcde)
(:domain blocksworld)
(:objects a b c d e)
(:init (on−table a) (clear a)
   (on−table b) (clear b)
   (on−table e) (clear e)
  (on−table c) (on d c) (clear d)
  (arm−empty))
(:goal (and (on e c) (on c a) (on b d))))
```

10.3 Planning Complexity

10.3.1 Satisficing planning

→ find a plan for Π or "unsolvable"

10.3.2 Optimal planning

→ find an optimal plan for Π or "unsolvable"

→

10.3.3 PlanEx

$\rightarrow$ problem of deciding, whether or not there exists a plan

$\rightarrow$ PSPACE-complete = PSPACE-hard + in PSPACE

10.3.4 PlanLen

$\rightarrow$ problem of deciding, whether or not there exists a plan of at most length X

$\rightarrow$ PSPACE-complete

10.3.5 PolyPlanLen

$\rightarrow$ problem of deciding, whether or not there exists a plan of at most length X, whereas X is bounded by a polynomial in the size of Π

$\rightarrow$ NP-complete

10.4 Relaxing in planning

- Problem class P with heuristic h_P
- Transformation R that transforms P to P'
- Simpler problem class P' with optimal heuristic $h^*_{P'}$

$\rightarrow$ $h_P = h^*_{P'}$

10.4.1 Delete Relaxation

R: When the world changes, its previous state remains true as well
$\Rightarrow$ Delete list of P' is empty

$PlanEx^+$ (deciding whether or not there exists a relaxed plan) is a member of P

10.4.2 h^+ heuristic

$h^+ \rightarrow$ ideal delete-relaxation heuristic

h^+ is admissible

h^+ is NP-hard to compute